Noah Buchanan
Problem Set 3
Distributed Systems

October 6, 2020

I implemented threading into the original calculate() method by simply creating a thread for each column and each thread goes down the columns calculating the masked value for each index in that column and using the same method to calculate the masked value from the non threaded version, the only difference is simply that k number of threads are doing that same method's logic(with some minor changes) at the same time. Each thread is limited to its own column and that way they can all work simultaneously and no more than k threads will be running at any time.

I originally had plans of creating a thread for every single index including row and column, but still limiting it to k threads running at the same time and it would work by calling .join() on the thread that is 1 above the current thread in the column right at the beginning of run(). This way I could have multiple threads ready to go once the prior one finishes and if one thread is bottle necked the rest can still finish on time but I did not have enough time to implement something like this but essentially it would work the same way just with new threads on each row in the columns as well as some minor changes that I haven't realized I imagine.

ThreadingLab2 Class:

```
public class ThreadingLab2 {

        /*******************************
        Name: Noah Buchanan
        Username: dist103
        Problem Set: PS3
        Due Date: October 6, 2020
        *******************************/


        public static class UAThread implements Runnable {
```

```
        public int threadId;
        public int j;
        public int[][] A;
        public int[][] B;
        public int[][] C;
        public int buffer;

        public UAThread(int threadId, int j, int[][] A, int[][] B, int[][] C, in

                this.threadId = threadId;
                this.j = j;
                this.A = A;
                this.B = B;
                this.C = C;
                this.buffer = buffer;
        }

        @Override
        public void run() {

                int bx = 0;
                int by = 0;
                for (int i = 0; i < A.length; i++) {
                        bx = 0;
                        for (int x = i - buffer; x <= i + buffer; x++) {
                                by = 0;
                                for (int y = j - buffer; y <= j + buffer; y++) {

                                        if (x >= 0 && y >= 0 && x < A.length &&

                                                C[i][j] += A[x][y] * B[bx][by];
                                        }
                                        by++;
                                }
                                bx++;
                        }
                }
        }

}



public static int[][] calculate2(int[][] A, int[][] B) throws Exception {
```

```
        int buffer = (int) B.length / 2;
        int[][] C = new int[A.length][A[0].length];
        int threadId = 0;
        Thread[] threads = new Thread[A[0].length];
        // int j = 0;
        for (int j = 0; j < A[0].length; j++) {
                Thread t = new Thread(new UAThread(threadId, j, A, B, C, buffer)
                threads[j] = t;
                threads[j].start();
                threadId++;
        }
        for (int i = 0; i < threads.length; i++) {
                threads[i].join();
        }

        return C;
}
```

```
public static int[][] calculate(int[][] A, int[][] B) {

        int buffer = (int) B.length / 2;
        int[][] C = new int[A.length][A[0].length];
        int one;
        int two;

        for (int i = 0; i < A.length; i++) {

                for (int j = 0; j < A[0].length; j++) {
                        one = 0;
                        for (int x = i - buffer; x <= i + buffer; x++) {
                                two = 0;
                                for (int y = j - buffer; y <= j + buffer; y++) {

                                        if (x >= 0 && y >= 0 && x < A.length &&
                                                C[i][j] += (A[x][y] * B[one][two
                                        }
                                        two++;
                                }
                                one++;
```

```
                                        }
                                }
                        }

                        return C;
                }


}
```